第4章 数据抽象

本章内容

- 通过比较用C和C++实现的数组（数据容器）类，说明面向对象的特点。
- 如何在C++中定义类。

数据抽象

- 抽象的优点
  - 运行效率；
  - 安全性；
  - 可维护性；
- 库的使用
  - 最好有源代码

一个袖珍的C库

- 需要这样一个工具，它类似一个数组，但它的长度能在运行时根据需要调整。
- 我们首先需要在头文件中定义该“数组”存储的内容：

```
typedef struct CStashTag {
    int size;      // Size of each space
    int quantity;  // Number of storage spaces
    int next;      // Next empty space
    // Dynamically allocated array of bytes:
    unsigned char* storage;
} CStash;
```

一个袖珍的C库

- 在头文件中，另外还需要定义一组函数对该结构进行处理：

```
void initialize(CStash* s, int size);
void cleanup(CStash* s);
int add(CStash* s, const void* element);
void* fetch(CStash* s, int index);
int count(CStash* s);
void inflate(CStash* s, int increase);
```

一个袖珍的C库

- 下面的代码是我们将如何使用这个C库的示例：保存100个整数，并将其取出，输出。

```
//ClibTest.cpp
CStash intStash;
 int i;
 // Now remember to initialize the variables:
 initialize(&intStash, sizeof(int));
 for(i = 0; i < 100; i++)
     add(&intStash, &i);
 for(i = 0; i < count(&intStash); i++)
     cout << "fetch(&intStash, " << i << ") = "
          << *(int*)fetch(&intStash, i)  //强制类型转换
          << endl;
 cleanup(&intStash);
```

一个袖珍的C库

```
CStash stringStash;
char* cp;
ifstream in;
string line;
const int bufsize = 80;
// Holds 80-character strings:
initialize(&stringStash, sizeof(char)*bufsize);
in.open("CLibTest.cpp");
assert(in);
while(getline(in, line))
    add(&stringStash, line.c_str());
i = 0;
while((cp = (char*)fetch(&stringStash,i++))!=0)
    cout << "fetch(&stringStash, " << i << ") = "
         << cp << endl;
```

实现C库

- 初始化：

```
// Quantity of elements to add
// when increasing storage:
const int increment = 100;

void initialize(CStash* s, int sz) {
  s->size = sz;
  s->quantity = 0;
  s->storage = 0;
  s->next = 0;
}
```

实现C库

- 在数组中添加一个元素：

```
int add(CStash* s, const void* element) {
   if(s->next >= s->quantity) //Enough space left?
           inflate(s, increment);
    // Copy element into storage,
    // starting at next empty space:
   int startBytes = s->next * s->size;
   unsigned char* e = (unsigned char*)element;
   for(int i = 0; i < s->size; i++)
           s->storage[startBytes + i] = e[i];
    s->next++;
   return(s->next - 1); // Index number
}
```

实现C库

- 从“数组”中获取一个元素

```
void* fetch(CStash* s, int index) {
    // Check index boundaries:
    assert(0 <= index);
    if(index >= s->next)
        return 0; // To indicate the end
     // Produce pointer to desired element:
    return &(s->storage[index * s->size]);
    //return s->storage + index * s->size;
}
```

实现C库

- 获取“数组”中元素的个数

```
int count(CStash* s) {
   return s->next;  // Elements in CStash
}
```

实现C库

- 增加内存

```
void inflate(CStash* s, int increase) {
   assert(increase > 0);
   int newQuantity = s->quantity + increase;
   int newBytes = newQuantity * s->size;
   int oldBytes = s->quantity * s->size;
   unsigned char* b = new unsigned char[newBytes];
   for(int i = 0; i < oldBytes; i++)
       b[i] = s->storage[i]; // Copy old to new
   delete [](s->storage); // Old storage
   s->storage = b; // Point to new memory
   s->quantity = newQuantity;
}
```

实现C库

- 清理

```
void cleanup(CStash* s) {
  if(s->storage != 0) {
    cout << "freeing storage" << endl;
    delete []s->storage;
  }
}
```

Dynamic storage allocation

- Allocate and release memory:
- C:
  - void *malloc(size_t size);
  - void *calloc(size_t numElements,size_t sizeOfElement);
  - void * realloc(void * p,int n);
  - void free (void * p);
- C++
  - new
  - delete

What's wrong

- Compare with Java equivalent
  - Name clashes
  - Information hidden

C++ Library

- The basic object：

```
struct Stash {
  int size;      // Size of each space
  int quantity;  // Number of storage spaces
  int next;      // Next empty space

  // Dynamically allocated array of bytes:
  unsigned char* storage;
  // Functions!
  void initialize(int size);
  void cleanup();
  int add(const void* element);
  void* fetch(int index);
  int count();
  void inflate(int increase);
}; ///:~
```

Using C++ Lib

Add and fetch integers

```
Stash intStash;
 intStash.initialize(sizeof(int));
 for(int i = 0; i < 100; i++)
    intStash.add(&i);
 for(int j = 0; j < intStash.count(); j++)
    cout << "intStash.fetch(" << j << ") = "
        << *(int*)intStash.fetch(j)
        << endl;
```

Compare： add(&intStash, &i) ➔intStash.add(&i)

Implement C++ Lib

- Scope
- this: implicit parameter

```
void Stash::initialize(int sz) {
    size = sz;
    quantity = 0;
    storage = 0;
    next = 0;
}
```

Size of object

- The size of a struct is the combined size of all of its members.
- Consider the size of following struct

```
struct A {
  int i[100];
};

struct B {
  void f();
};
```

Abstract data typing

- The ability to package data with functions allows you to create a new data type. This is often called encapsulation
- Stash is an abstract data type(user defined types), and can be used as int
- float/int have characteristics and behavior, so they are ADT (Aabstract Data Type) too.
- Message and Operation
  - object.memberFunction(arglist)
  - Object::memberFunction(arglist){}

Header file etiquette

- The header file is where the interface specification is stored.
- ensure a consistent declaration across the whole system
- ensure that the declaration and the definition match by including the header in the definition file
- The header file provides only information to the compiler but nothing that allocates storage by generating code or creating variables.
- separate the interface (the declaration) from the implementation (the definition of the member functions) so the implementation can be changed without forcing a re-compile of the entire system.

Header file etiquette

- 多次声明问题
  - 允许重声明函数，不允许重声明结构
- In each header file that contains a structure, you should first check to see if this header has already been included in this particular cpp file:

```
#ifndef HEADER_FLAG
#define HEADER_FLAG
……
#endif
```

- don’t put using directives in header files

Nested structures

- You can nest a structure within another structure, and therefore keep associated elements together.
- A Stack sample about memory allocate and release

```
int main(int argc, char* argv[]) {
        requireArgs(argc, 1); // File name is argument
        ifstream in(argv[1]);
        assure(in, argv[1]);
        Stack textlines;
        textlines.initialize();
        string line;
        // Read file and store lines in the Stack:
        while(getline(in, line))
                  textlines.push(new string(line));
                // Pop the lines from the Stack and print them:
        string* s;
        while((s = (string*)textlines.pop()) != 0) {
                cout << *s << endl;
                delete s;
        }
        textlines.cleanup();
} ///:~
```

Global scope resolution

```
// Global scope
// resolution

int a;
void f( ) { }

class S
{
   int a;
   void f( );
};
```

```
void S::f( )
{
   ::f();        // global f();
   ::a++;        // global a
   a--;          // struct's a
}
int main( )
{
    S s;
    f();         // global f();
}
```